SVEUČILIŠTE U SPLITU

SVEUČILIŠNI ODJEL ZA STRUČNE STUDIJE

Stručni diplomski studij Primijenjeno računarstvo

Crypto-project: Prikaz blockchaina i vizualizacija podataka vezanih uz rudarenje

Kriptovalute

Student: Anamarija Papić **Datum:** 15.1.2025.

Sadržaj

Opis projekta i korištene tehnologije

Crypto-project je web aplikacija koja omogućava pregled blokova u Bitcoin ili Litecoin blockchainu (ovisno o konfiguraciji) od posljednje potvrđenih prema prethodnima te vizualizaciju dobivenih podataka na grafovima.

Za razvoj aplikacije korišten je Next.js razvojni okvir za React JavaScript koji se koristi za izradu *full-stack* web aplikacija. Arhitektura Next.js-a omogućava izvršavanje i poslužiteljskog i klijentskog koda, što ga čini dobrim izborom za ovaj projekt. Radi se o jednostraničnoj aplikaciji (*single web page* – SPA).

Next.js projekt kreiran je i postavljen koristeći naredbu `npx create-next-app@latest`. Za *frontend* sučelje korišten je Tailwind CSS.

Paketi su instalirani koristeći npm upravitelj paketa. Za Bitcoin RPC API pozive korištena je biblioteka `bitcoin-core`. Za WebSocket podršku instalirani su paketi `socket.io` i `socket.io-client`, za podršku pri vizualizaciji grafova paketi `chart.js` i `react-chartjs-2`, a za podršku pri formatiranju i prikazivanju vrijednosti veličine i vremena biblioteke `prettyBytes` i `fromnow`.

Također je implementirano keširanje (engl. *caching*), za što je korišten Redis. Keširanje omogućava da se već prethodno dohvaćeni podatci o blokovima ne moraju ponovno dohvaćati s RPC API-ja, nego ih se dohvaća iz *cachea*. Redis je pokrenut u Dockeru na lokalnom računalu naredbom `docker run -p 6379:6379 -it redis/redis-stack-server:latest`, a se na njega iz aplikacije pristupa putem Redis klijenta iz npm paketa `redis` koji je instaliran za tu potrebu.

Za dohvaćanje povijesnih podataka o cijeni kriptovalute (bitcoin ili litecoin, ovisno o konfiguraciji) i njezine protuvrijednosti u američkim dolarima (USD) korišten je vanjski javni API koji nudi mempool.space[1] odnosno litecoinspace.org[2].

[1] https://mempool.space/docs/api/rest
[2] https://litecoinspace.org/docs/api/rest

Nakon instalacije potrebnih paketa naredbom `npm install`, aplikacija se pokreće naredbom `npm run start`. Dostupna je u web pregledniku na ruti http://localhost:3000/blocks.

Sljedeća komponenta na stranici je horizontalno skrolabilna, a prikazuje dohvaćene blokove od najnovije potvrđenog prema prethodnima te relevantne podatke.

Zatim slijede vizualni prikazi na grafovima. Prikazani grafovi vezani su uz statistike o rudarenju: prikaz raspona naknada kroz vrijeme, prikaz omjera nagrade u bloku i naknade kroz vrijeme te prikaz vrijednosti naknada u bitcoinu ili USD protuvrijednosti izrudarenog bloka također kroz vremenski period.

Sučelje

U ovom poglavlju sadržane su slike zaslona (engl. *screenshots*) izrađene web aplikacije.

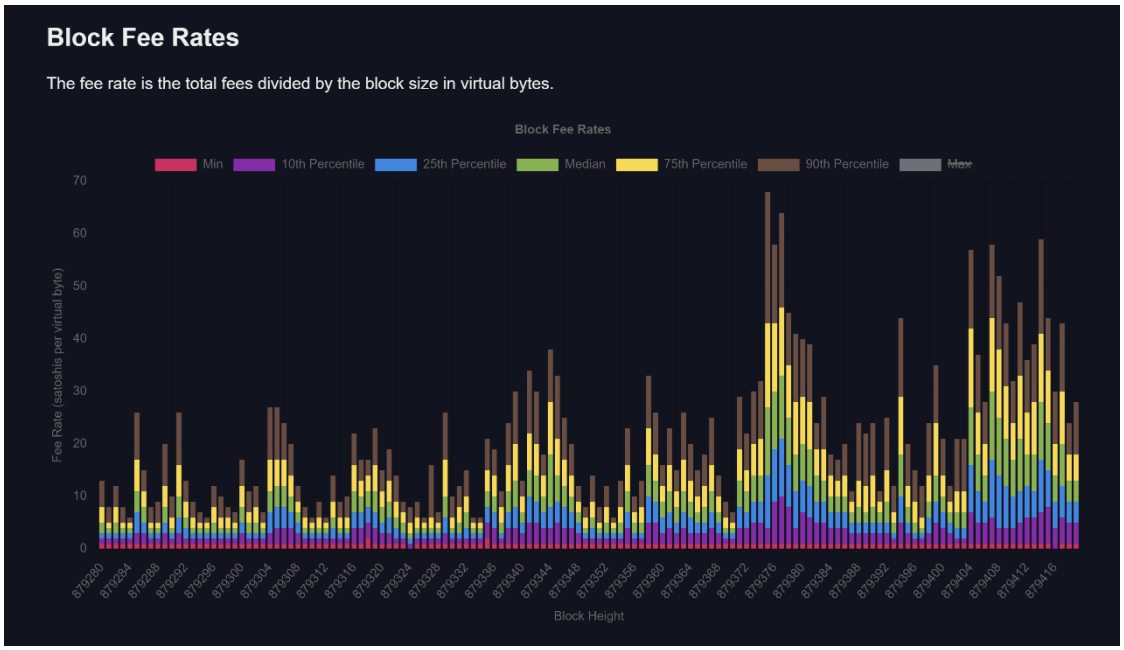
Kako je vidljivo na slici 1, na samom vrhu stranice prikazan je izbornik s vremenskim intervalima (24H, 3D, 1W, 1M) za koji želimo dohvatiti blokove iz blockchaina. Sljedeća komponenta na stranici je horizontalno skrolabilna, a prikazuje dohvaćene blokove od najnovije potvrđenog prema prethodnima. Prikazani podatci za blok su: visina, hash, ukupni broj transakcija u bloku, vrijeme bloka prikazano u *human-readable* obliku koje se ažurira s protokom vremena, težina, raspon naknada, medijan naknada, prosječna naknada, ukupna naknada, ukupna nagrada, ukupna vrijednost bloka, adresa rudara za isplatu nagrade izvučena iz coinbase transakcije te cijena kriptovalute u američkim dolarima u trenutku objave u blockchain.

Slika 1: Izbornik s vremenskim intervalima te prikaz blockchaina u realnom vremenu

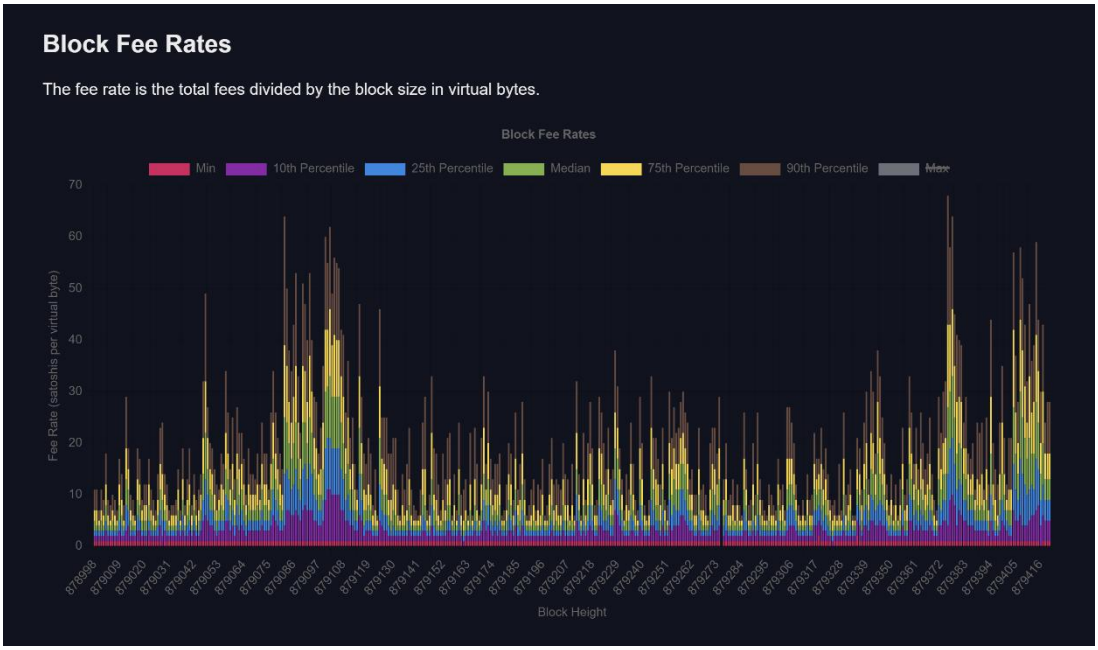
Novi blokovi koji se pojave odmah su prikazani zahvaljujući inicijaliziranom WebSocketu, jer isti osluškuje rutu za dohvaćanje trenutnog *best block hash* tj. posljednjeg bloka te ako je *best block hash* različit od u hasha posljednje dohvaćenog bloka u aplikaciji odašilje signal na koji se prikaz blockchaina u aplikaciji ažurira.

Potom slijede vizualizacije podataka vezanih uz rudarenje.

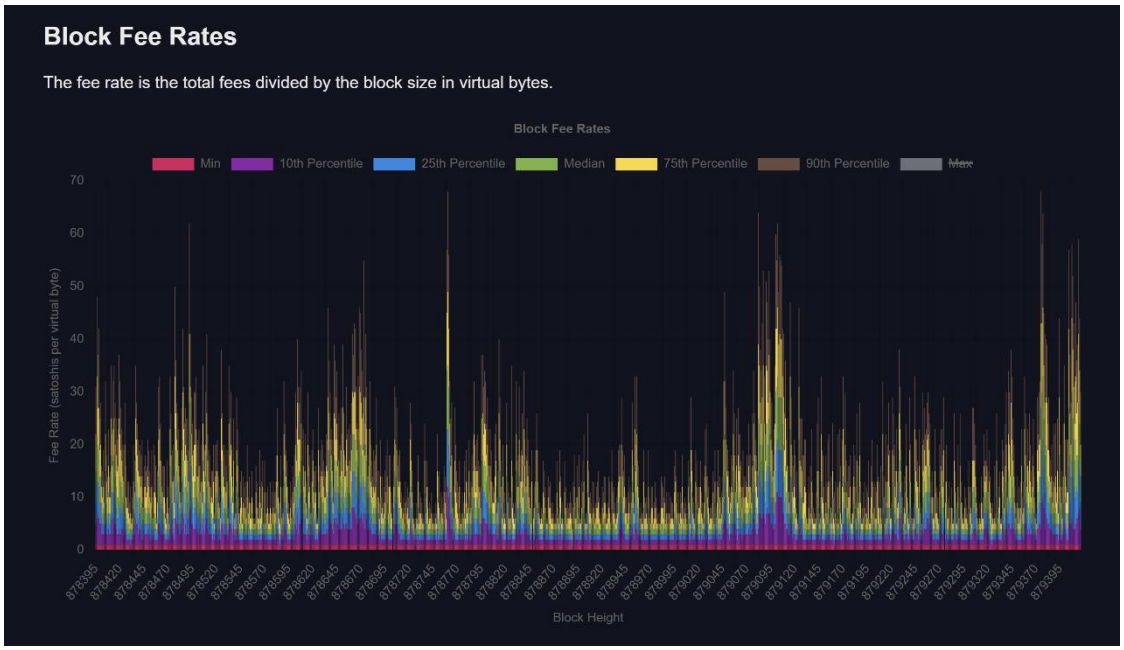
Prvi prikazani grafikon vizualizira raspone naknada u bloku kroz vrijeme (vidljivo prema visini bloka te na legendi koja se pokazuje prelaskom miša preko grafikona. Na slici 2 prikazani su podatci za 24 sata tj. 1 dan, na slici 3 za 3 dana, a na slici 4 za 1 tjedan.



Slika 2: Prikaz raspona naknada kroz vrijeme (24H)

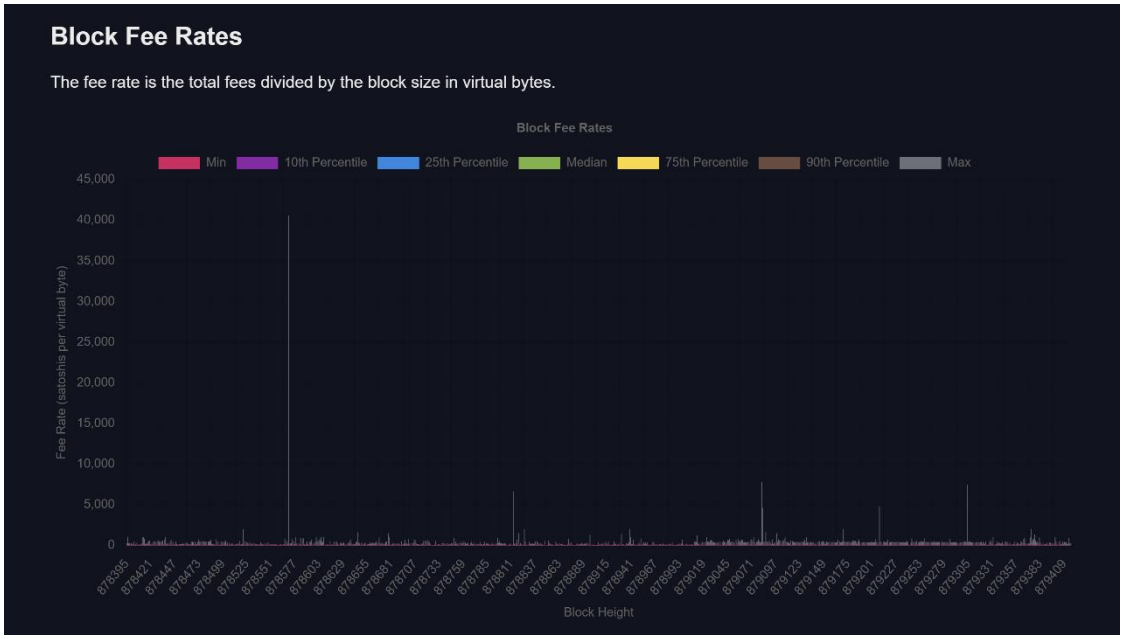


Slika 3: Prikaz raspona naknada kroz vrijeme (3D)



Slika 4: Prikaz raspona naknada kroz vrijeme (1W)

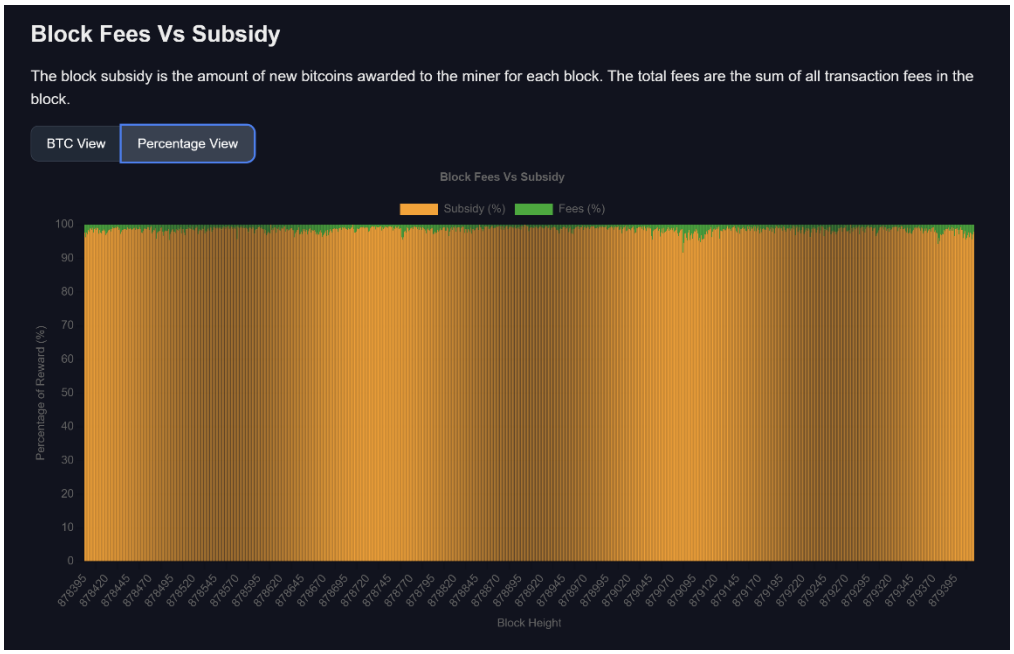
Na istom se grafu može prikazati i maksimalne naknade što vidimo na slici 5 za vremenski raspon od 1 tjedna.



Slika 5: Prikaz maksimalnih naknada kroz vrijeme (1W)

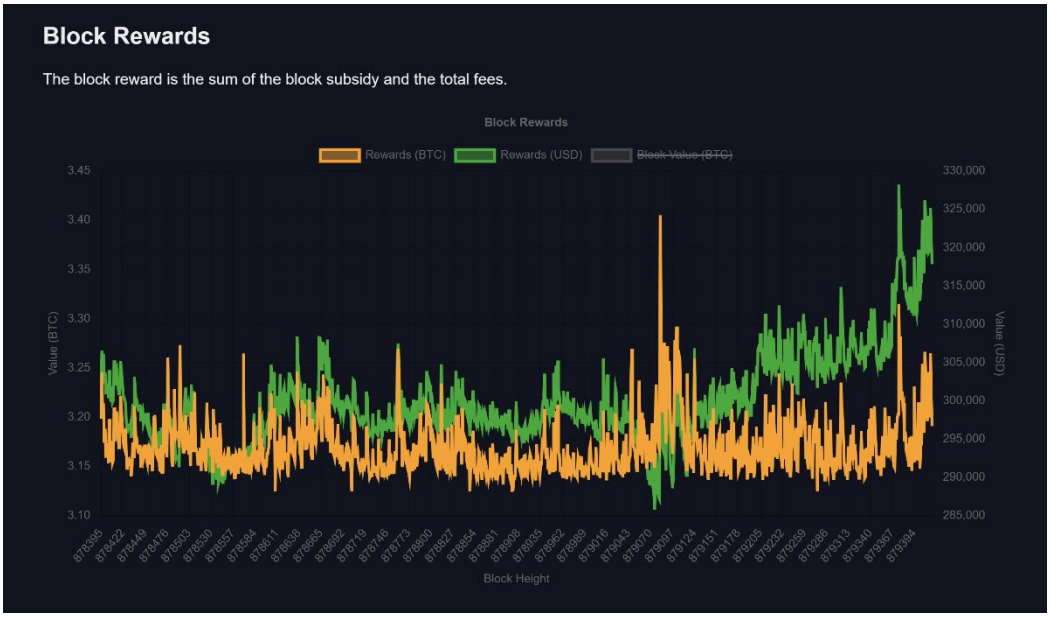
Na slikama 6 i 7, prikazan je graf omjera naknade i nagrade u bloku, prvo kao vrijednost u kriptovaluti, a na drugoj slici kao postotna vrijednost.

Slika 6: Vizualizacija omjera naknade i nagrade u bloku (1W)

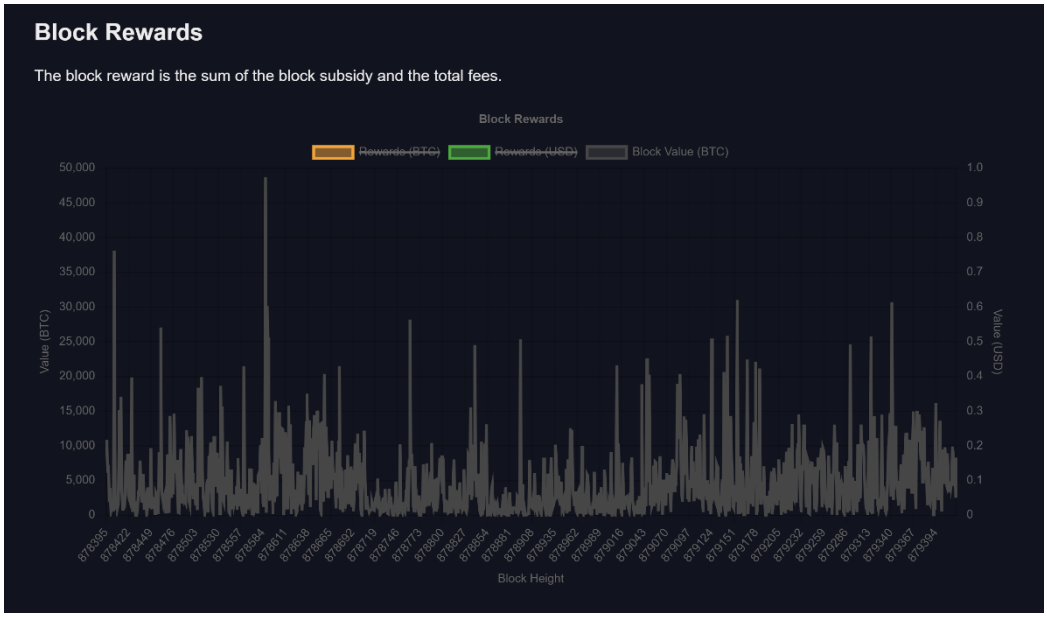


Slika 7: Vizualizacija omjera naknade i nagrade u bloku kao postotna vrijednost (1W)

Na slici 8 prikazan je posljednji grafikon – vizualizacija naknada u izrudarenim blokovima kroz vremenski period (ovdje 1 tjedan) izražena u kriptovaluti i USD protuvrijednosti. Prikaz grafa se također može promijeniti tako da prikazuje ukupne vrijednosti blokova izražene u kriptovaluti u tom intervalu, što je vidljivo na slici 9.



Slika 8: Prikaz vrijednosti naknada u kriptovaluti i USD protuvrijednosti (1W)



Slika 9: Prikaz ukupne vrijednosti bloka izražene u kriptovaluti (1W)

Popis korištenih API poziva

Tablica 1: Popis korištenih poziva prema sučelju Bitcoin Core

Poziv	**Funkcija**
`getBlockCount`	Vraća visinu najopterećenijeg potpuno validiranog lanca tj. najrecentniji blok.
`getBestBlockHash`	Vraća hash najboljeg bloka u najopterećenijem potpuno validiranom lancu tj. najrecentniji blok.
`getBlock`	Korišten sa zastavicom verbosity=1, vraća objekt s informacijama o određenom bloku.
`getBlockHash`	Vraća hash bloka u najboljem blockchainu na pruženoj visini.
`getBlockStats`	Vraća statistiku za blok. Svi iznosi su u satoshijima.
`getRawTransaction`	Vraća sirove podatke transakcije. Korišten sa zastavicom verbose=true.

Izvadci iz programskog koda

U ovom poglavlju bit će izdvojeni najvažniji dijelovi kôda aplikacije. U ispisu 1 prikazana je inicijalizacija Bitcoin RPC API klijenta. Varijable okruženja iščitavaju se iz `.env` datoteke, a vrijednosti se mogu postaviti tako da se ostvari spajanje na Bitcoin ili Litecoin mainnet mrežu.

```
import Client from 'bitcoin-core';

const client = new Client({
  host: process.env.RPC_HOST,
  port: process.env.RPC_PORT,
  username: process.env.RPC_USER,
  password: process.env.RPC_PASS,
});

module.exports = client;
```

Ispis 1: Inicijalizacija klijenta za pristup Bitcoin RPC API-ju

U ispisu 2 prikazane su pomoćne funkcije sa poslužiteljskog dijela aplikacije koje pozivaju Bitcoin RPC API.

```
export const getBlockCount = async () => {
  return bitcoinClient.getBlockCount();
};

export const getBestBlockHash = async () => {
  return bitcoinClient.getBestBlockHash();
};

export const getBlock = async (blockHash) => {
  return bitcoinClient.getBlock(blockHash);
};

export const getBlockHash = async (blockHeight) => {
  return bitcoinClient.getBlockHash(blockHeight);
};

export const getBlockStats = async (blockHash) => {
  return bitcoinClient.getBlockStats(blockHash);
};

export const getRawTransaction = async (txid, verbose = true, blockHash) =>
{
  return bitcoinClient.getRawTransaction(txid, verbose, blockHash);
};
```

Ispis 2: Funkcije za dohvat podataka s Bitcoin RPC API-ja

U ispisu 3 prikazane su funkcije zaslužne za keširanje: inicijalizacija Redis klijenta te fukcije za dohvaćanje podataka iz *cachea* ili spremanje u *cache*.

```
import { createClient } from 'redis';

const redisClient = createClient().on('error', (err) =>
  console.log('Redis Client Error', err)
);

await redisClient.connect();

export const setBlockData = async (unit, hash, data) => {
  const key = `${unit}:${hash}`;
  await redisClient.set(key, JSON.stringify(data));
};

export const getBlockData = async (unit, hash) => {
  const key = `${unit}:${hash}`;
  const data = await redisClient.get(key);
  return data ? JSON.parse(data) : null;
};

export default redisClient;
```

Ispis 3: Funkcije vezane uz keširanje

Na sljedećim ispisima (4 i 5) prikazan je način na koji je realizirano dohvaćanje blokova u određenom vremenskom intervalu, što je zasigurno najvažnija funkcionalnost u aplikaciji. Zbog velikog broja rezultata, uvedena je paginacija na način da se zadano dohvaća po 10 rezultata.

```
export const getTimeRangeBlocks = async (timeRange, page = 1, limit = 10)
=> {
  const now = Math.floor(Date.now() / 1000);
  let startTime;

  switch (timeRange) {
    case '24h':
      startTime = now - 24 * 60 * 60;
      break;
    case '3d':
      startTime = now - 3 * 24 * 60 * 60;
      break;
    case '1w':
      startTime = now - 7 * 24 * 60 * 60;
```

```
      break;
    case '1m':
      startTime = now - 30 * 24 * 60 * 60;
      break;
    default:
      startTime = now - 24 * 60 * 60;
  }

  const blockCount = await getBlockCount();
  const start = (page - 1) * limit;
  const end = start + limit;

  const blocks = await fetchBlocks(
    blockCount - start,
    blockCount - end,
    startTime
  );

  return blocks;
};
```

Ispis 4: Funkcija `getTimeRangeBlocks()`

```
export const fetchBlocks = async (start, end, startTime) => {
  const unit = getCryptoUnit();
  const blocks = [];

  for (let i = start; i > end && i > 0; i--) {
    const blockHash = await getBlockHash(i);
    const cachedBlock = await getBlockData(unit, blockHash);
    if (cachedBlock) {
      if (cachedBlock.time < startTime) break;
      blocks.unshift(cachedBlock);
    } else {
      const block = await getBlock(blockHash);
      if (block.time < startTime) break;
      const blockStats = await getBlockStats(blockHash);

      // Extract the coinbase transaction
      const coinbaseTx = block.tx[0];
      const miner = await getMinerFromCoinbase(coinbaseTx, blockHash);

      // Get historical price data
      const apiUrlBase =
        unit === 'LTC'
          ? 'https://litecoinspace.org/api/v1'
          : 'https://mempool.space/api/v1';
```

```
      const priceResponse = await fetch(
        `${apiUrlBase}/historical-
price?currency=USD&timestamp=${block.time}`
      );
      const priceData = await priceResponse.json();

      const blockData = {
        ...blockStats,
        ...block,
        miner,
        price: priceData?.prices?.[0]?.USD || 0,
      };

      await setBlockData(unit, blockHash, blockData);
      blocks.unshift(blockData);
    }
  }

  return blocks;
};
```

Ispis 5: Funkcija `fetchBlocks()`

U ispisu 6 prikazano je dohvaćanje coinbase transakcije – prve transakcije u bloku koja na svome izlazu sadrži podatke za isplatu rudaru koji je izrudario pojedini blok. U funkciji se dohvaća adresa rudara na koju će se isplatiti nagrada za izrudareni blok.

```
export const getMinerFromCoinbase = async (coinbaseTx, blockHash) => {
  const tx = await getRawTransaction(coinbaseTx, true, blockHash);
  const miner =
    tx?.vout?.[0]?.scriptPubKey?.address ||
    tx?.vout?.[0]?.scriptPubKey?.addresses?.[0] ||
    'Unknown';
  return miner;
};
```

Ispis 6: Funkcija `getMinerFromCoinbase()`

U ispisu 7 prikazana je inicijalizacija WebSocketa na poslužiteljskoj strani aplikacije te implementirano da svakih 60 sekundi šalje provjeru je li izrudaren neki novi blok, te ako jest dohvaća njegove podatke (što je već prethodno prikazano) i odašilje ga.

```
const SocketHandler = (req, res) => {
  if (!res.socket.server.io) {
    console.log('Socket is initializing');
    const io = new Server(res.socket.server);
    res.socket.server.io = io;
```

```
    io.on('connection', (socket) => {
      console.log('Client connected');

      let lastBlockHash = null;

      const sendNewBlock = async () => {
        const bestBlockHash = await getBestBlockHash();

        if (bestBlockHash !== lastBlockHash) {
          // Get block data
          socket.emit('newBlock', {
            ...block,
            ...blockStats,
            miner,
            price: priceData?.prices?.[0]?.USD || 0,
          });
          lastBlockHash = bestBlockHash;
        }
      };

      // Send new blocks to the client
      setInterval(sendNewBlock, 60000); // Check for new blocks every 60
seconds

      socket.on('disconnect', () => {
        console.log('Client disconnected');
      });
    });
  } else {
    console.log('Socket is already running');
  }
  res.end();
};

export default SocketHandler;
```

Ispis 7: Funkcija `SocketHandler()`

U ispisu 8 prikazana je React `useEffect` kuka (engl. *hook*) na klijentskoj strani aplikacije koja će primiti odaslani blok i ažurirati prikaz blockchaina.

```
useEffect(() => {
    const socket = io();

    socket.on('newBlock', (newBlock) => {
      setBlocks((prevBlocks) => {
        if (!prevBlocks.some((block) => block.hash === newBlock.hash)) {
```

```
          return [...prevBlocks, newBlock];
        }
        return prevBlocks;
      });
    });

    return () => {
      socket.disconnect();
    };
  }, []);
```

Ispis 8: `useEffect` kuka na klijentskoj strani aplikacije za primanje novih blokova